

Highly Efficient Entropy Extraction for True Random Number Generators on FPGAs

Vladimir Rozic¹, Bohan Yang¹, Wim Dehaene², Ingrid Verbauwhede¹

¹ESAT/COSIC and iMinds, KU Leuven

²ESAT/MICAS, KU Leuven

Kasteelpark Arenberg 10

B-3001 Leuven-Heverlee, Belgium

{Vladimir.Rozic,Bohan.Yang,Wim.Dehaene,Ingrid.Verbauwhede}@esat.kuleuven.be

ABSTRACT

True random number generators are essential components in cryptographic hardware. In this work, a novel entropy extraction method is used to improve throughput of jitter-based true random number generators on FPGA. By utilizing ultra-fast carry-logic primitives available on most commercial FPGAs, we have improved the efficiency of the entropy extraction, thereby increasing the throughput, while maintaining a compact implementation. Design steps and techniques are illustrated on an example of a ring-oscillator based true random number generator on Spartan-6 FPGA. In this design, the required accumulation time is reduced by 3 orders of magnitude compared to the most efficient oscillator-based TRNG on the same FPGA. The presented implementation occupies only 67 slices, achieves a throughput of 14.3 Mbps and it is provided with a formal evaluation of security.

Categories and Subject Descriptors

SEC1.3 [Hardware Security]: Device, circuit and architecture techniques for security

General Terms

Security, Design

Keywords

Random Number Generation, Randomness Extraction

1. INTRODUCTION

Random number generators (RNG) are crucial components in every security system. They are used for generating session keys, challenges and padding values. A true random number generator (TRNG) is a type of RNG that generates unpredictable bits, as opposed to deterministic or pseudo-random number generator (DRNG or PRNG) which

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

DAC '15, June 07 - 11, 2015, San Francisco, California USA

Copyright 2015 ACM 978-1-4503-3520-1/15/06 ...\$15.00

<http://dx.doi.org/10.1145/2744769.2744852>

only provides statistically strong but completely predictable bit sequences. Since RNGs are used in sensitive applications such as key generation and challenge-response based authentication, ensuring the unpredictability of the generated bits is of paramount importance. Due to the sensitivity of crypto applications and the important role of random numbers in security protocols, TRNGs are subject of strict evaluation. While it is relatively easy to invent circuits that generate seemingly random bits, a formal evaluation of security is required today. This evaluation requires not only a rigorous statistical testing of the output, but also a mathematical model of the randomness source and an assessment of entropy. FPGA implementations are especially challenging because of the limited variety of components and routing resources.

Early TRNG implementations on FPGAs were not provided with entropy models. The proposed generators were verified using batteries of statistical tests such as NIST [7] and DIEHARD [3] but they lack formal evaluation of security. Only a handful of TRNG designs on FPGAs is provided with mathematical models and entropy estimations.

A design with multiple ring oscillators was proposed in [9] and a high-throughput implementation on Virtex2 FPGA was presented in [8]. However, the underlying assumptions behind the proof of security have been challenged [2]. In addition, the design consumes too much FPGA resources because it requires 110 ring oscillators. A compact TRNG with a security proof, implemented on Xilinx Spartan-3E FPGA was presented in [11], but this design achieves a throughput of only 250kb/s. Design based on self-timed ring-oscillators was presented in [1]. This design achieves 100Mb/s throughput and it is accompanied by an entropy model but it consumes too much resources on FPGA since it requires a self-timed ring oscillator with 511 stages.

Our novel approach relies on the efficient entropy extraction from the accumulated timing jitter. Rather than boosting throughput by increasing the number of transition events at the cost of high energy and/or area, we focus on improving the entropy extraction from a single transition event. Our main idea is based on the fact that the TRNG throughput increases proportionally to the square of the timing resolution of the sampled signal. For this reason, even a modest improvement in the timing resolution leads to high reduction in accumulation time and the proportional throughput gain.

On Xilinx FPGAs, two types of slices contain a carry-chain primitive which can be used to generate branches,

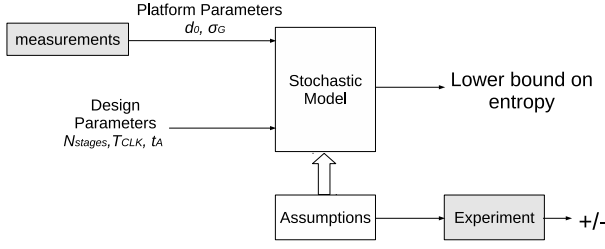


Figure 1: TRNG Design and Evaluation procedure.

adders or multipliers. This primitive consists of four delay stages connected to flip-flops and can be configured as a tapped delay line. Carry chains from the neighboring slices on the same column can be connected through dedicated routing paths to form a larger delay line. This configuration can be used for sampling signals with a high timing accuracy and it has been used for implementing high-resolution time-to-digital converters [6]. By utilizing carry chains for entropy extraction, our TRNG implementation achieved a compact footprint with high-throughput. A security evaluation of this TRNG is provided.

2. TRNG EVALUATION METHOD

The early approach for TRNG evaluation consisted of collecting the random data and running a set of statistical tests such as NIST and DIEHARD batteries of test. A major flaw of this black box approach is the fact that all pseudo-random number generators can easily pass the statistical tests even though they produce completely deterministic output. This approach is problematic even when used by the TRNG designer for verifying his own design. For example, a designer may believe that the randomness is caused by the thermal jitter when in fact it is coming from the unstable power supply. In that case, if the TRNG is used with a voltage stabilizer it may produce very weak keys.

While statistical tests can be used for the initial analysis or a sanity check, they are not a replacement for a formal security evaluation. Formal security evaluation of a TRNG requires a mathematical model of the TRNG, and an entropy assessment. AIS-31 [5] provides a framework for design and evaluation of TRNGs. One of the requirements for AIS-31 certification is the stochastic model of the entropy source and the estimation of the lower boundary of the generated entropy.

Stochastic model is used to describe a probabilistic system that evolves over time. It is more simple than the physical model (such as transistor model) because it only takes into account the processes that affect the randomness. This type of model is used to describe the process of generating entropy in the circuit, for example jitter accumulation or a process of resolving from a metastable state.

Figure 1 shows the block diagram for a TRNG evaluation. A stochastic model has to be based on clearly stated and experimentally verifiable assumptions. The goal of the model is to provide the entropy estimation for different values of platform parameters (such as LUT delays or the noise parameters) and design parameters (such as the number of ring-oscillators or the sampling frequency). The obtained stochastic model computes the minimal entropy based on the input parameters (platform and design parameters). The

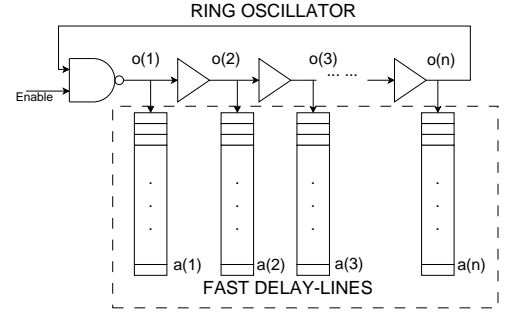


Figure 2: General Architecture.

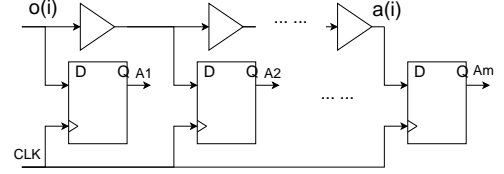


Figure 3: Fast delay line with taps.

next phase in the design flow is obtaining the platform parameters by experiment. Afterwards, optimal design parameters need to be determined. This can be done by computing the entropy for different values of the design parameters, using the stochastic model and the measured values of the platform parameters. Design parameters can then be tuned until the specified entropy bound is reached. Statistical analysis is done at the last stage of the evaluation.

3. PROPOSED ARCHITECTURE

Our architecture is based on the concept of sampling the unstable signal with high precision, thereby extracting more entropy compared to the standard approach. This method is very effective because gain in terms of throughput scales with the square of the improvement in precision. In this architecture, the entropy source and the entropy extractor are implemented as separate blocks. All randomness in the system is generated by the timing uncertainty of the signal edge. Oscillator signals are first propagated through the tapped delay lines, a sample is taken and the position of the signal edge is determined from the captured data snippet. Digitized data are propagated to the entropy extractor which then generates one random bit at the output.

Entropy source and the digitization block are shown in Figure 2. Entropy source is implemented as a free-running n -stage oscillator using a single NAND gate and several buffers. Digitization block consists of fast, tapped delay lines which are connected to the output of each delay element. These delay lines are performing time-to-digital conversion of the noisy signals.

Figure 3 shows the internal structure of a single tapped delay line. The line consists of m fast buffers and a set of flip-flops connected to their outputs. Timing resolution of the conversion is equal to the delay of a single stage of the tapped delay line. After the rising clock edge, output data of each buffer is captured in flip-flops. This data reflects the internal state of the oscillator. In order to capture the full state of the oscillator, the parameter m , which is the size of

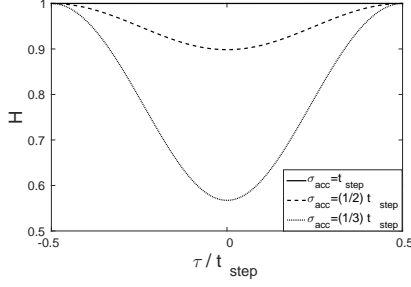


Figure 7: Shannon entropy depending on τ for different values of the accumulated jitter.

each delay element are propagated through the fast delay lines. Since the oscillator is free-running, the jitter accumulates over time. As long as the delay lines are long enough ($m \cdot t_{step} > d_{0,LUT}$) it is guaranteed that at least one noisy signal edge is captured in the delay lines. Figure 6 shows the model of the entropy extraction. Relative thermal jitter between the sampling edge and the noisy signal is described using Gaussian distribution. Since the jitter realizations are independent, standard deviation of the accumulated thermal jitter after time t_A is proportional to the square root of the number of transition events:

$$\sigma_{acc}(t_A) = \sigma_{LUT} \sqrt{\frac{t_A}{d_{0,LUT}}} \quad (1)$$

The noisy signal is sampled with precision t_{step} and neighboring states of the TDC are encoded using different bits. Probabilities of one and zero are equal to the area under the curve.

These probabilities depend on the offset time t_o between the sampling edge and the most likely position of the noisy signal edge as shown in the Figure 6. We define τ as the interval between the noisy signal edge and the middle of the closest sampling bin as shown in figure:

$$\tau = (t_o \bmod t_{step}) + \frac{t_{step}}{2} \quad (2)$$

Without the loss of generality, we can assume that this bin is decoded as 1. Then the binary probability is given by:

$$P_1 \approx \sum_{i=-\infty}^{+\infty} \left(\Phi\left(\frac{\tau - (2 \cdot i - \frac{1}{2})t_{step}}{\sigma_{acc}(t_A)}\right) - \Phi\left(\frac{\tau - (2 \cdot i + \frac{1}{2})t_{step}}{\sigma_{acc}(t_A)}\right) \right) \quad (3)$$

where Φ is the cumulative probability function of the Gaussian distribution:

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-\frac{t^2}{2}} dt \quad (4)$$

4.3 Lower Bound of Entropy

The binary probabilities depend on the offset time τ . The exact value of τ depends on the accumulation time, the number of sample in the sequence, but also on the low frequency noise and deterministic noise. Since these factors are not predictable nor controllable, the worst case value should be

used to estimate the lower bound for entropy. Shannon entropy is then given by:

$$H = -P_1 \cdot \log_2(P_1) - (1 - P_1) \cdot \log_2(1 - P_1) \quad (5)$$

Figure 7 shows the Shannon entropy depending on τ . Lower bound is reached for $\tau = 0$.

4.4 Use of the Model

The presented stochastic model can be used to estimate the lower bound of entropy from the platform parameters and the design parameters. This model consists of a set of equations which can be implemented as a Matlab function that produces a lower bound on entropy given the platform and design parameters. Platform parameters are the physical parameters of the implementation platform and they should be determined by measurements. Relevant platform parameters for this design are: $d_{0,LUT}$ - average delay of a single LUT, t_{step} -bin width of the fast delay lines, and σ_{LUT} -thermal noise generated by a single transition event.

Once the platform parameters are known, the model is used to determine the lower bound of entropy for different design parameter values. The design parameters are: n -number of stages in a ring oscillator, m -number of stages in the fast delay line, f_{CLK} -system clock frequency, t_A - jitter accumulation time (it can also be expressed as N_A - the number of system clock periods), and an optional down-sampling factor k . Down-sampling can be used to improve the linearity of the time-to-digital conversion in the fast delay lines by combining k neighboring bins into a single bin. The stochastic model is used to determine the entropy for different values of the design parameters, which enables exploring different design trade-offs.

Therefore, the design procedure of a TRNG consists of the following four steps:

Step1: Measuring relevant platform parameters.

Step2: Determining optimal design parameters based on the stochastic model and the obtained platform parameters.

Step3: FPGA Implementation.

Step4: Statistical evaluation of the generated bits.

4.5 Post-processing

Generated bits can be improved using post-processing. Post-processing is a compression technique which increases the entropy-per-bit at the price of reduced throughput. Xor post-processing is a simple method with compact hardware implementation. It consists of xoring n_p consecutive generated bits, thereby decreasing throughput by a factor of n_p . Presented stochastic model can be used to estimate the maximal bias of the generated numbers, defined as:

$$b = \max(P_1, P_0) - 0.5 \quad (6)$$

The bias of the post-processed sequence with compression rate n_p is then:

$$b_{pp} = 2^{n_p-1} \cdot b^{n_p} \quad (7)$$

The new entropy value can then be computed.

5. IMPLEMENTATION

A Xilinx Spartan-6 FPGA was used for implementation. Implementation method of the entropy source and digitizer

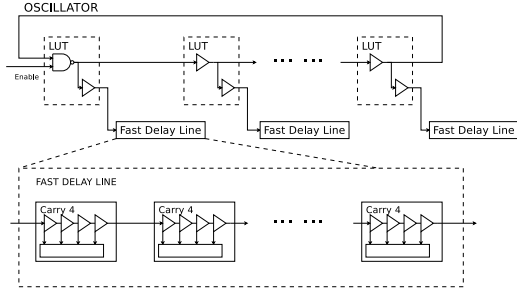


Figure 8: Entropy source implementation on Xilinx Spartan-6 FPGA.

is shown in Figure 8. Stages of the ring-oscillator are implemented using LUTs, and fast delay lines are implemented using carry-chain primitives. On Spartan-6, one half of the slices contains these carry primitives. These slices are located in even numbered columns. Long carry chains are formed by connecting carry primitives from the neighboring slices in the same column. Delay stages of the oscillator are placed in slices directly below the fast delay lines. These are the only placement constraints that we used in our implementation. The rest of the design was synthesized and implemented automatically.

5.1 Platform parameters

Platform parameters of interest are the LUT delays $d_{0,LUT}$, standard deviation of the thermal jitter $\sigma_{G,LUT}$ and the time step of the TDC conversion t_{step} , i.e. the delay of a single element in a carry-chain.

LUT delays are determined by implementing a ring oscillator, and counting the number of transitions within a fixed time period. It was found that $d_{0,LUT} = 480ps$.

Tapped-line delay step was determined by capturing an oscillator output in a long carry chain, and counting the number of stages of a clock period. The result is approximately $t_{step} = 17ps$.

Thermal jitter measurement has to be implemented very carefully because this parameter is of critical importance. Historically, there have been many papers that overestimated this parameter, and there are several challenges to overcome in order to get the accurate result. The most reliable way to make the measurements is on-chip because thermal noise can be filtered out on the pin, package and the scope. According to circuit models presented in [4], measurement time needs to be very short (order of $1\mu s$ or shorter), otherwise low frequency noise becomes dominant. The measurements have to be carried out differentially in order to account for the global noise sources, such as instability in the power supply. We present a simple method to determine jitter on FPGAs using carry-chain logic. Two identical ring-oscillators are implemented and placed close to each other. The oscillators are enabled for $20ns$, and the output is captured using fast, tapped delay lines based on carry4 primitives. The captured data is then sent to the PC for the analysis. Accumulated jitter is determined by observing the difference between the signal edges of the two oscillators. Standard deviation of the jitter is estimated from 1000 measurement results. Obtained results show that $\sigma_{G,LUT} \approx 2ps$.

5.2 Design decisions

The number of ring-oscillator stages n , doesn't figure in the entropy model. In order to achieve the most compact implementation, the value of this parameter should be chosen to be as small as possible. We chose the value $n = 3$ because this was the shortest ring-oscillator for which we could reliably measure the frequency and the jitter parameters.

The number of stages in the fast-delay lines m has to be chosen such that the signal edge is always detected, which happens for $m > d_{0,LUT}/t_{step}$. For our platform parameters this condition becomes $m > 29$. Since each carry4 primitive has 4 elements, m has to be a multiple of 4. Initially, we tried using 8 carry primitives ($m = 32$) but the measurements showed that the signal edge is not captured in 0.8% of the cases. This is probably due to the fact that d_0 is the average delay value and some LUTs may be slower. In order to provide more robustness, we have decided to use 9 carry4 stages ($m = 36$) and the measurements showed that the edge is always captured. There is no benefit from further increasing m .

One of the challenges to overcome, is the non-linearity of the carry-chain, i.e. different bins have different widths. A study conducted in [6] showed that the main reason for this non-linearity is the unbalanced clock tree. The same paper also suggests to improve the linearity by using carry chains that span across only one clock region. On Spartan-6, clock regions spans across 16 rows. Since our design uses only 9 carry4 stages it is possible to set placement constraints to ensure that all 9 stages are in the same clock region. Another reason for the non-linearity of the time-to-digital conversion is the internal structure of the carry4 primitive as well as the influence of the process variations. This can be improved by using down-sampling by a factor of $k = 4$ which results in wider bins and higher requirements for the t_A . In order to explore the design space, we implemented several versions of the TRNG, using both $k = 1$ and $k = 4$.

Accumulation time t_A has to be a multiple of $10ns$, because the platform clock frequency is $100MHz$. We explore the design space for different values of t_A for both versions of the TRNG.

5.3 Comparison with elementary TRNG

Using the stochastic model and the obtained platform parameters we compare our entropy extraction method with the one used by an elementary TRNG on the same platform. Elementary TRNG consists of a free-running oscillator sampled by a system clock. Jitter accumulation process is exactly the same as described in our model, but the entropy extraction is different since the noisy signal is sampled with timing-precision equal to the half-period of the ring oscillator. In the best case scenario, ring oscillator is implemented using only one LUT, which would result in $t_{step,RO} = d_{0,LUT}$.

Since the throughput scales proportionally to the square of the sampling precision, the throughput improvement that we obtain for $k = 1$ is:

$$\frac{d_{0,LUT}^2}{t_{step}^2} = 797 \quad (8)$$

which is almost 3 orders of magnitude.

For $k = 4$ the improvement factor is 49.8.

Table 1: Evaluation of different design versions. H_{RAW} -Shannon entropy of the raw bits, n_{NIST} -required compression rate needed to pass all NIST tests, H_{NEW} -entropy after post-processing.

k	$t_A[ns]$	H_{RAW}	n_{NIST}	H_{NEW}	Throughput [Mb/s]
1	10	0.99	7	0.999	14.3
	20	0.999	7	0.999	7.14
4	10	0.03	> 16	NA	NA
	50	0.7	13	0.999	1.53
	100	0.94	10	0.999	1
	200	0.99	6	0.999	0.83

Table 2: Comparison with related work.

Work	Platform	Resources	Throughput [Mb/s]
[8]	Virtex 2 pro	565 slices	2.5
[1]	Cyclone 3	>511 LUTs	133
	Virtex 5	>511 LUTs	100
[11]	Spartan 3E	Not reported	0.25
This work ($k=1$)	Spartan 6	67	14.3
This work ($k=4$)	Spartan 6	40	1.53

6. RESULTS

We have implemented two versions of the TRNG on Xilinx Spartan-6 FPGA. Parameter values are $n = 3$ and $m = 36$ for both versions. One version uses downsampling ($k = 4$) and the other one doesn't ($k = 1$). TRNG for $k = 1$ occupies 67 slices including the entropy source and the entropy extractor. TRNG version with $k = 4$ is more compact as it occupies only 40 slices.

Table 1 shows the statistical evaluation and throughput results for different design parameters. Generated data were compressed using xor post-processing, n_{NIST} is the minimal compression rate needed to pass all statistical tests. Entropy per bit before and after compression (H_{RAW} and H_{NEW}) was calculated from the model. Throughput after compression is reported in the last column.

Comparison with related work is summarized in Table 2. Our design achieves higher throughput than all TRNGs except [1]. However, the TRNG in [1] uses 511 LUTs just for the entropy source, which is at best 128 slices on Xilinx platforms (the exact utilization results are not reported in the paper). Our entropy source is a ring oscillator which consumes only 3 slices, and the complete design consumes 40 slices.

7. CONCLUSIONS AND FUTURE WORK

In this work, we present a novel entropy extraction technique for high-throughput, true random number generators on FPGAs. This technique relies on the carry-logic primitives for efficient sampling of the accumulated jitter. The technique was illustrated on an example of a high-throughput TRNG implemented on Xilinx Spartan-6 FPGA. All design and evaluation steps were shown, including the platform pa-

rameter measurements, exploring the design space and running the statistical evaluation. Fastest TRNG implementation occupies 67 slices and achieves throughput of 14.3Mb/s. The most compact implementation consumes 40 slices and achieves throughput of 1.53Mb/s.

Future work will focus on applying the presented methodology on different implementation platforms as well as developing embedded tests for on-the-fly evaluation.

8. ACKNOWLEDGMENTS

This work was supported in part by the Research Council KU Leuven: GOA TENSE (GOA/11/007). In addition, this work is supported in part by the Flemish Government through FWO G.0550.12N, G.0130.13N and FWO G.0876.14N, the Hercules Foundation AKUL/11/19, and by a grant from Intel. In addition, this work was supported in part by the Scholarship from China Scholarship Council (No.201206210295).

9. REFERENCES

- [1] A. Cherkaoui, V. Fischer, L. Fesquet, and A. Aubert. A very high speed true random number generator with entropy assessment. In *CHES'13*, volume 8086 of *LNCS*, pages 179–196. Springer, 2013.
- [2] M. Dichtl and J. D. Golić. High-speed true random number generation with logic gates only. In *CHES'07*, pages 45–62, Berlin, Heidelberg, 2007. Springer-Verlag.
- [3] DIEHARD battery of tests of randomness.
- [4] P. Haddad, Y. Teglia, F. Bernard, and V. Fischer. On the assumption of mutual independence of jitter realizations in P-TRNG stochastic models. In *DATE 2014*, pages 1–6. IEEE, 2014.
- [5] W. Killmann and W. Schindler. A proposal for: Functionality classes for random number generators. BDI, Bonn, 2011.
- [6] H. Menninga, C. Favi, M. Fishburn, and E. Charbon. A multi-channel, 10ps resolution, FPGA-based TDC with 300MS/s throughput for open-source PET applications. In *Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC), 2011 IEEE*, pages 1515–1522, Oct 2011.
- [7] A. Rukhin et al. A statistical test suite for random and pseudorandom number generators for cryptographic applications. SP 800-22 NIST, August 2008.
- [8] D. Schellekens, B. Preneel, and I. Verbauwhede. FPGA vendor agnostic true random number generator. In *FPL*, pages 1–6. IEEE, 2006.
- [9] B. Sunar, W. Martin, and D. Stinson. A provably secure true random number generator with built-in tolerance to active attacks. *Computers, IEEE Transactions on*, 56(1):109–119, jan. 2007.
- [10] B. Valtchanov, A. Aubert, F. Bernard, and V. Fischer. Modeling and observing the jitter in ring oscillators implemented in FPGAs. In *DDECS 2008*, pages 1–6, april 2008.
- [11] M. Varchola and M. Drutarovsky. New High Entropy Element for FPGA Based True Random Number Generators. In *CHES 2010*, volume 6225 of *LNCS*, pages 351–365. 2010.